Getting started with the FPGA demo bundle for Xilinx

Xillybus Ltd.

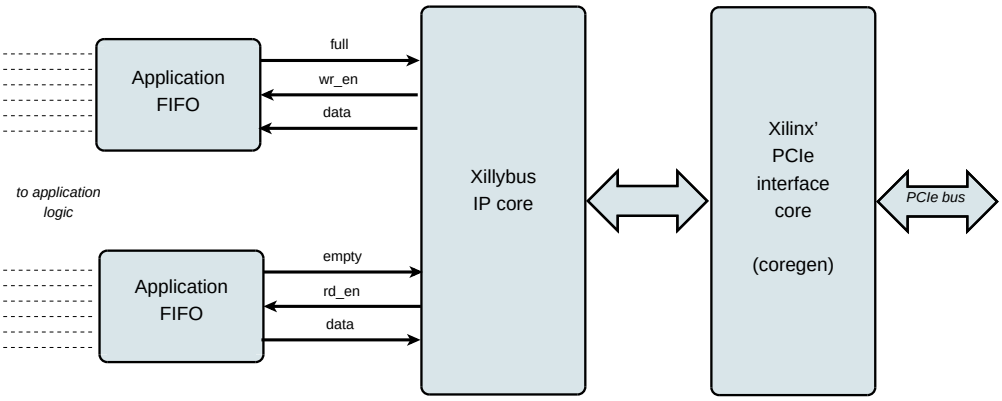
www.xillybus.com

Version 3.3

1

Introduction

Xillybus is a DMA-based end-to-end solution for data transport between an FPGA and a host that runs Linux or Microsoft Windows. It offers a simple and intuitive interface to the designer of the FPGA logic as well as to the programmer of the software.



As shown above, the application logic on the FPGA only needs to interact with standard FIFOs.

For example, writing data to the lower FIFO in the diagram makes the Xillybus IP core sense that data is available for transmission in the FIFO's other end. Soon, the IP core reads the data from the FIFO and sends it to the host, making it readable by the userspace software. The data transport mechanism is transparent to the application logic in the FPGA, which merely interacts with the FIFO.

On its other side, the Xillybus IP core implements the data flow utilizing PCI Express' Transport Layer level, generating and receiving TLP packets. For the lower layers, it relies on Xilinx' official PCIe core, which is part of the development tools, and requires no additional license (even when using the WebPACK edition).

The application on the computer interacts with device files that behave like named pipes. The Xillybus IP core and driver transport data efficiently and intuitively between the FIFOs in the FPGAs and their related device files on the host.

With XillyUSB, an MGT transceiver is used to implement an USB 3.0 interface, which is used for data transport instead of the PCIe interface mentioned above.

The IP core is built instantly per customer's spec, using an online web application. The number of streams, their direction and other attributes are defined by customer to achieve an optimal balance between bandwidth performance, synchronization, and simplicity of design. After going through the preparation steps with the demo bundle, as described in this guide, it's recommended to build and download your custom IP core at http://xillybus.com/custom-ip-factory.

This guide explains how to rapidly set up the FPGA with a Xillybus IP core, which can be attached to user-supplied data sources and data consumers, for real application scenario testing. The IP core is included in a demo bundle, which can be downloaded at the website.

Despite its name, the demo bundle is not a demonstration kit, but a fully functional starter design, which can perform useful tasks as is.

For those who are curious, a brief explanation on how Xillybus is implemented can be found in Appendix A of either Xillybus host application programming guide for Linux or Xillybus host application programming guide for Windows.

2

Prerequisites

2.1 Hardware

The Xillybus FPGA demo bundle is packaged to work with several boards and devices, as listed on the download pages (see section 2.2 below).

Owners of other boards may run a demo bundle on their own hardware after making the necessary changes in pin placements and verifying that the MGT's reference clock is handled properly. This should be straightforward to any fairly experienced FPGA engineer. More about this in section 4.3.

2.2 FPGA project

The Xillybus demo bundle is available for download at Xillybus site's download pages. For the PCIe-based cores:

http://xillybus.com/pcie-download

And for XillyUSB:

http://xillybus.com/usb-download

The demo bundle includes a specific configuration of the Xillybus IP core, which is intended for simple tests. Therefore, it has a relatively poor performance for certain applications.

Custom IP cores can be configured, automatically built and downloaded using the IP Core Factory web application. Please visit http://xillybus.com/custom-ip-factory for using this tool.

Any downloaded bundle, including the Xillybus IP core, is free for use, as long as this

use reasonably matches the term “evaluation”. This includes incorporating the core in end-user designs, running real-life data and field testing. There is no limitation on how the core is used, as long as the sole purpose of this use is to evaluate its capabilities and fitness for a certain application.

2.3 Development software

The recommended tool for the implementation of Xillybus’ demo bundle (as well as other designs involving Xillybus) is listed below, depending on the FPGA’s family.

Xillybus for PCIe:

As of today, it’s almost certain that the Vivado version that is installed on your computer is suitable for working with Xillybus for PCIe. Nevertheless, these are the requirements in more detail:

- When using Virtex-5 FPGAs, the Xilinx ISE 13.1 version is preferred, see paragraph 4.3.5.
- For Spartan-6 and Virtex-6, use Xilinx ISE 13.2 and later.
- For Kintex-7 and Virtex-7 with Gen2 interface (all Virtex-7 that aren’t XT/HT, and 485T too), Vivado 2014.1 and later is the preferred tool. Among the ISE revisions, version 14.2 and later is recommended.
- For Virtex-7 with Gen3 interface (XT/HT except 485T), Vivado 2014.1 and later is preferred. If ISE is chosen, revision 14.6 and later is required.
- For Artix-7, Vivado 2014.1 and later should be used. ISE 14.6 and later is fine as well.
- For Kintex / Virtex Ultrascale, Vivado 2015.2 and later should be used. No ISE revision supports these devices.
- Ultrascale+ FPGAs require Vivado 2017.3 and later.
- Versal APAC FPGAs require Vivado 2021.2 and later.

XillyUSB:

- For all FPGAs except Ultrascale+, Vivado 2015.2 and later should be used.
- For Ultrascale+, Vivado 2018.3 and later should be used.

This software can be downloaded directly from Xilinx' website (http://www.xilinx.com).

Any of this software's editions is suitable. If the FPGA is covered by the WebPACK Edition, this edition may be the preferred choice, as it can be downloaded and used with no license fee for an unlimited time.

The implementation of Xillybus relies on some IP cores that are supplied by Xilinx. All software editions cover these IP cores, without any need for additional licensing.

2.4 Experience with FPGA design

When the design is intended for a board that appears in the list of demo bundles, no previous experience with FPGA design is necessary to have the demo bundle working on the FPGA. When other boards are used, it's required to have some knowledge with using Xilinx' tools, in particular defining pin placements and clocks.

To make the most of the demo bundle, a good understanding of logic design techniques, as well as mastering an HDL language (Verilog or VHDL) are necessary. Nevertheless, the Xillybus demo bundle is a good starting point for learning these, as it presents a simple starter design to experiment with.

3

The implementation of the demo bundle

3.1 Overview

For Vivado users who want to skip reading everything below, these are the steps for running the first implementation:

- Uncompress the demo bundle to a working directory.
- Start Vivado, or close any open project if Vivado is already running.
- Pick Tools > Run Tcl Script... and choose **xillydemo-vivado.tcl** in verilog/ or vhdl/ (inside the demo bundle).
- Click “Generate Bitstream” (or “Generate Device Image”).
- After a successful implementation, find the bitstream file in vivado/xillydemo.runs/impl_1/.

And now to the longer story: There are three possible methods for the implementation of Xillybus’ demo bundle, and obtaining a bit stream file:

- Using the project files in the bundle as they are. This is the simplest way, and is suitable when working with boards that appear in the list of demo bundles, except for ML506 (Virtex-5).
- Modifying the files to match a different FPGA. This is suitable when working with other boards, and/or other FPGAs. This is also necessary when working with Virtex-5. More information about this in paragraph 4.3.
- Setting up the Vivado (or ISE) projects from scratch. Possibly necessary when integrating the demo bundle with existing application logic. Further details in paragraph 4.2.

In the remainder of this section, the first work procedure is detailed, which is the simplest and most commonly chosen one. The other two work procedures are based upon the first one, with differences that are detailed in the paragraphs given above.

IMPORTANT:

The evaluation bundle is configured for simplicity rather than performance. Significantly better results can be achieved for applications requiring a sustained and continuous data flow, in particular for high-bandwidth cases. For these scenarios, a custom IP core is easily built and downloaded with the web application.

3.2 File outline

The bundle consists of some of these directories (which directories are present depends on the intended FPGA):

- core – The Xillybus IP core is stored here
- instantiation templates – Contains the instantiation templates for the core (in Verilog and VHDL)
- verilog – Contains the project file for the demo bundle and the sources in Verilog (in the 'src' subdirectory)
- vhdl – Contains the project file for the demo bundle and the sources in VHDL (in the 'src' subdirectory)
- vivado-essentials – Definition files and build directories for logic, for use by Vivado.
- blockplus – This directory is relevant only for Virtex-5. See paragraph 4.3.5.

Note that each demo bundle is intended for a specific board, as listed at the site's web page from which the demo bundle was downloaded. If another board is used, or if certain configuration resistors have been added or removed from the board, the constraints file must be edited accordingly.

For Vivado projects, this file is vivado-essentials/xillydemo.xdc, and for ISE projects it's the UCF file in the chosen 'src' directory under verilog/ or vhdl/.

Also note that the vhdl directory contains Verilog files, but none of these should need significant changes.

The interface between Xillybus' IP core and the application logic takes place in the xillydemo.v file or xillydemo.vhd file (in the respective 'src' subdirectories). This is the file to edit in order to try Xillybus with your own data.

3.3 Generating the bitstream file with Vivado

ISE users: Please skip to paragraph 3.4.

Vivado generates many intermediate files in a relatively complex structure, which makes it difficult to keep the project under control. In order to keep the file structure in the bundle compact, a script in Tcl is supplied for creating the Vivado project. This script creates a new subdirectory, "vivado", and populates this directory with files as necessary.

The project relies on the files in the src/ subdirectory (no copies of these files are made). The PCIe block, as well as the FIFOs that are used by the logic, are defined in vivado-essentials/. Vivado also populates this directory with intermediate files during the project's implementation.

Start Vivado. With no project open, Pick Tools > Run Tcl Script... and choose **xillydemo-vivado.tcl** in the verilog/ or vhdl/ subdirectory, depending on your preference. A sequence of events takes place during less than a minute. The success of the project's deployment can be verified by choosing the "Tcl Console" tab at Vivado's window's bottom, and verify that it says:

```
INFO: Project created: xillydemo
```

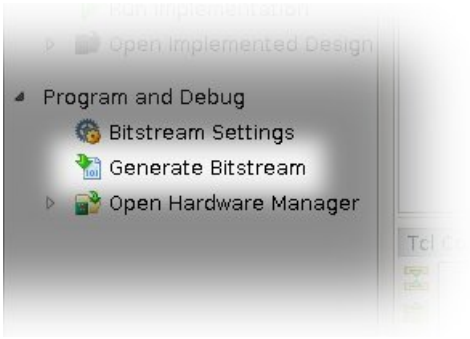
If this is not the last line in the Tcl console, something went wrong, most likely because the wrong revision of Vivado is used.

Warnings will appear during this stage, but no errors. However if the project has already been generated (i.e. the script has been run already), attempting to run the script again will result in the following error:

```
ERROR: [Common 17-53] User Exception: Project already exists on disk,
       please use '-force' option to overwrite:
```

Note that the new "vivado" subdirectory is created in the directory that contains the Tcl script.1

After the project has been created, start an implementation: Click "Generate Bitstream" (or "Generate Device Image") on the Flow Navigator bar to the left.



A popup window that asks if it's OK to launch synthesis and implementation is likely to appear – pick "Yes".

Vivado runs a sequence of processes. This takes a few minutes normally. Several warnings are issued, none of which are classified critical (but some critical warnings may still remain in the logs from the execution of the Tcl script).

A popup window will appear, which will say that the implementation of the bitstream was completed successfully. It will give choices of what to do next. Any option is fine, including "Cancel".

The bitstream file, xillydemo.bit, can be found at vivado/xillydemo.runs/impl_1/. For Versal FPGAs, the file is xillydemo.pdi instead.

The implementation is never expected to fail. There is however a one error condition worth mentioning:

An error saying "Timing constraints weren't met" can happen when custom logic has been integrated. This means that the tools failed to achieve the requirements for timing. In this case, the design is syntactically correct, but needs corrections to make certain paths fast enough with respect to given clock rates and/or I/O requirements. The process of correcting the design for better timing is often referred to as *timing closure*.

A timing constraint failure is commonly announced as a critical warning, so Vivado doesn't prevent the user from producing a bitstream file that doesn't guarantee the FPGA's reliable behavior. To prevent the creation of such a bitstream, a timing failure is turned into an error by virtue of a small Tcl script, namely "showstopper.tcl", which is automatically executed at the end of a route run. To turn this safety measure off, click "Project Settings" under "Project Manager" in the Flow Navigator. Choose the "Implementation" button, and scroll down to the settings for "route_design". Then remove showstopper.tcl from tcl.post.

Vivado users may skip the following sections, and go directly to paragraph 3.6.

3.4 Setting up Xilinx' PCIe IP core

This part relates only to the ISE toolchain, for FPGAs other than Spartan-6. If Vivado is used, please refer to paragraph 3.3. Those working with Spartan-6 FPGAs may jump directly to paragraph 3.5.

A somewhat peculiar organization of the Coregen IP core for PCIe doesn't allow the inclusion of the XCO file in the implementation project, but instead, the core generating software creates Verilog files for inclusion. This is the case only when working with Virtex-5, Virtex-6 and series-7.

Virtex-5 FPGAs require some special handling of the XCO files. See paragraph 4.3.5.

Please find the project file (.xise) in the blockplus directory or the pcie_core directory (as found in your bundle) and double-click it to open ISE. Under "Design Utilities", click "Regenerate all cores" and wait for the process to finish. Then just close ISE. There is no need for any further action.

This procedure generates a set of Verilog files, which are wrappers for the Xilinx PCIe core. These files are used by the project that creates the demo bundle's bitstream. The files are generated in Verilog, regarless of your choice of Verilog or VHDL as the preferred language.

These is no need to manually include these Verilog files in the main project, but these files have to be generated once before the implementation of the entire project is attempted. There is no need to repeat this procedure, even not before a repeated implementation of the main project.

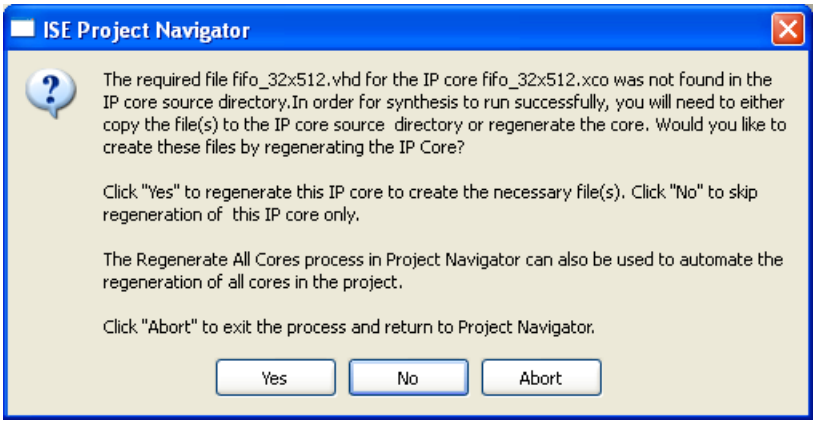
3.5 Generating the bit file with the ISE suite

When working with an FPGA that belongs to the Virtex or series-7 families, please make sure you've prepared the PCIe wrapper, as instructed in paragraph 3.4 above.

Depending on your preference, double-click the 'xillydemo.xise' file in either the 'verilog' subdirectory or the 'vhdl' subdirectory. ISE will start running and open the project with the correct settings. ISE shouldn't complain that any file is missing. If it does, and the FPGA family is one of series-7 or any Virtex family, it's likely that your PCIe wrapper wasn't prepared as mentioned in paragraph 3.4.

Click "Generate Programming File" to start the implementation.

During the first implementation, there is a need to regenerate two or three Xilinx Core-gen IP cores. This process is time-consuming, but fortunately it's done only once. A popup window looking something like this will appear two or three times:



Click "Yes" on all of these.

The procedure will produce several warnings (an implementation of an FPGA project always does) but should not present any errors. When the process is finished, the bitfile can be found as xillydemo.bit.

Always verify that the timing constraints were achieved by looking for the following sentence in the log (somewhere near the end):

```
All constraints were met.
```

This is crucial, in particular after making changes in the project. If the constraints aren't achieved, the tools will still generate a bitfile, but the FPGA may behave in an unpredictable manner (possibly as a result of temperature changes within the allowed temperature range).

A similar message can be found in ISE's design summary.

Failing to achieve timing constraints could be a result of adding logic that isn't fast enough to withstand the rate of bus_clk. But if a failure occurs for no apparent reason, it could be that Xilinx' tools made a poor initial guess when attempting to place the logic components on the FPGA's logic fabric, and the optimization algorithm running later on couldn't fix this.

The latter case can be fixed by changing the placer cost table figure (which is just a seed to the randomness of the initial placement). In the Processes pane inside ISE

Project Navigator , right-click “Map” and pick “Process Properties...”. Make sure that the Property display level is “Advanced” and change the “Starting Placer Cost Table” to just any other number that hasn’t been tried yet. The magnitude of this number has no significance. Then restart with “Generate Programming File”.

IMPORTANT:

On some ISE versions, notably ISE 14.2, the build in the verilog/ directory may fail with an error saying ERROR:HDLCompiler:687 - “C:/try/xillybus-eval-kintex7-1.1/verilog/src/fifo_32x512_synth.v” Line 54: Illegal redeclaration of module fifo_32x512. (or similar). This is due to a bug in Xilinx’ tools. To work around this, delete fifo_8x2048_synth.v and fifo_32x512_synth.v in the src/ directory, and restart “Generate Programming File”. Several warnings will indicate that the tools fail to find these files, but the implementation should nevertheless run through properly.

3.6 Loading the bitfile

In early development stages, it’s recommended to load the FPGA via JTAG. On most boards, a simple USB cable between the computer that runs Vivado and a USB connector on the board’s panel is enough. Those who work with ISE, use iMPACT for loading the bitfile.

Please refer to your board’s instructions on how to load the FPGA via JTAG.

For XillyUSB projects, the FPGA can be loaded and reloaded at any time, even while the USB interface is connected to a working computer.

With PCIe projects, the FPGA must be loaded with the bitfile before the computer is powered up: The computer expects the PCIe peripheral to be in a proper state when it powers up, and may not tolerate any surprises afterwards.

Therefore, do **not** reload the FPGA as long as the host is running. Even though the PCIe specification requires support for hotplugging, motherboards don’t normally expect a PCIe card to disappear and then reappear. Accordingly, some motherboards may not respond correctly. Nevertheless, reloading of the FPGA, while the operating system is running, works on *some* motherboards.

Xillybus’ driver is designed to respond sanely to hotplugging, however there is nothing to assure the computer’s general stability. This is dicussed on this page:

http://xillybus.com/doc/hot-reconfiguration

If the FPGA powers up and is loaded from a flash memory along with powering up the computer, it's essential to ensure that the FPGA is loaded quickly enough, so the PCIe device is present when the BIOS scans the bus.

4

Modifications

4.1 Integration with custom logic

The Xillybus demo bundle is constructed for easy integration with application logic. The place for connecting data is the xillydemo.v or xillydemo.vhd file (depending on the preferred language). All other HDL files in the bundle can be ignored for the purpose of using the Xillybus IP core for transporting data between the host (Linux or Windows) and the FPGA.

Additional HDL files with custom logic designs may be added to the project that was prepared as described in paragraph 3.5 or 3.3, and then rebuilt by clicking “Generate Programming File” or “Generate Bitstream”. There is no need to repeat the other steps of the initial deployment, so the development cycle for logic is fairly quick and simple.

When attaching the Xillybus IP core to custom application logic, it is warmly recommended to interact with the Xillybus IP core only through FIFOs, and not attempt to mimic their behavior with logic, at least not in the first stage.

An exception for this is when connecting memories or register arrays to Xillybus, in which case the example that is shown in the xillydemo module should be followed.

In the xillydemo module, FIFOs are used to perform a data loopback. In other words, the data that arrives from the host is sent back to it. Both of the FIFO’s sides are connected to the Xillybus IP core, so the core is both the source of the data and the consumer of the data.

In a real-life usage scenario, only one of the FIFO’s sides is connected to the Xillybus IP core. The FIFO’s other side is connected to application logic, which supplies or consumes data.

The FIFOs that are used in the xillydemo module work with only one common clock for both sides, as both sides are driven by Xillybus' main clock. In a real-life application, it may be desirable to replace them with FIFOs that have separate clocks for reading and writing. This allows to driving the data sources and data consumers with a clock other than bus_clk. By doing this, the FIFOs serve not just as mediators, but also for proper clock domain crossing.

Note that the Xillybus IP core expects a *plain* FIFO interface, (as opposed to First Word Fall Through, FWFT) for streams from the FPGA to the host.

The following documents are related to integrating custom logic:

- The API for logic design: Xillybus FPGA designer's guide
- Getting started with Xillybus on a Linux host
- Getting started with Xillybus on a Windows host
- Xillybus host application programming guide for Linux
- Xillybus host application programming guide for Windows
- The guide to defining a custom Xillybus IP core

4.2 Inclusion in a custom project

If desired, it's possible to include the Xillybus IP core in an existing Vivado / ISE project, or create a new project from scratch.

If the project doesn't exist already, start a new project, and set it up as based upon your preferred HDL language and intended FPGA.

To include the Xillybus IP core in a Vivado project, it's recommended to edit xillydemo-vivado.tcl to reflect the custom project's source files and settings, and create a fresh project by running this script.

To include the Xillybus IP core in an ISE project:

- When working with FPGAs belonging to the Virtex-5/6 or series-7 families, the PCIe wrapper files need to be generated separately, as detailed in paragraph 3.4 (and also paragraph 4.3.5 for Virtex-5 FPGAs). The generated Verilog files should be added in the custom project (but not the XCO file).
- Add all files in one of the two src/ subdirectories (depending on your language preference) into the project.

- Add a directory to the Macro search path: In the process menu, under “Implementation”, right-click “Translate” and choose “Process Properties...”. Add the ’core’ subdirectory to the Macro Search Path property (browsing with the button to the far right). Failing to set this property will make the Translate stage to fail during the implementation, because it won’t find the xillybus_core.ngc file.
- If the xillydemo module isn’t the top level module of the projects, connect its ports to the top level.
- To attach the Xillybus IP core to custom application logic, edit the xillydemo module, replacing the existing application logic with the desired one.

4.3 Using other boards

4.3.1 General

When working with a board which doesn’t appear in the list of demo bundles, some slight modifications in the bundle are necessary.

The core generates a few GPIO LED outputs. It’s recommended to connect these to LEDs on the board, if there are any such vacant.

4.3.2 Using Xillybus for PCIe

Most purchased boards have their own example of an FPGA design, which shows how the PCIe interface is used on that board. It’s often easiest to locate the relevant pin assignments in the intended board’s XDC / UCF file, and modify the pins’ names to those that are used in Xillybus’ XDC / UCF file. Then it’s possible to replace the relevant rows in the XDC / UCF file that is used in Xillybus’ project.

The details about how to place the pins are given below.

Note that the most common mistake is with the reference clock of the PCIe bus. Connecting just any clock with the same frequency will not work: The tiny frequency difference between the motherboard’s clock and any other clock will make the transceiver lose lock sporadically, resulting in unreliable communication, and possibly a failure to detect the FPGA as a PCIe device.

Since the Xillybus core is based upon Xilinx’ PCIe core, Xilinx’ user guide is a valid source for considerations that are related to PCIe’s physical layer.

4.3.3 Working with Spartan-6 PCIe boards

For Spartan-6, the Xillybus core interfaces with the host through a PCIe bus port, which consists of 7 physical wires as follows:

- A pair of differential wires for the reference clock, with the names PCIE_250M_P and PCIE_250M_N: A clock with a frequency of 125 MHz (despite the net's name), which is derived from the PCIe bus clock, is expected on these wires. If a different clock is applied, the Xilinx PCIe Coregen core (defined by pcie.xco in the bundle) must be reconfigured to expect the real clock frequency. Additionally, a timing constraint must be updated, so that the TS_PCIE_CLK specification reflects the change.
- The host's master bus reset on PCIE_PERST_B_LS
- Serial data input pair, PCIE_RX0_P and PCIE_RX0_N
- Serial data output pair, PCIE_TX0_P and PCIE_TX0_N

These pins' assignments are set according to the board's wiring.

4.3.4 Working with Virtex-6 PCIe boards

For Virtex-6 the wiring is similar:

- A pair of differential wires for the reference clock, with the names PCIE_REFCLK_P and PCIE_REFCLK_N: A clock with a frequency of 250 MHz, which is derived from the PCIe bus clock, is expected on these wires. If a different clock is applied, the Xilinx PCIe Coregen core (defined by pcie_v6_4x.xco in the bundle) must be reconfigured to expect the real clock frequency. Such a change may also involve changes in the constraints. Please refer to the example UCF file, generated by Coregen.
- The host's master bus reset on PCIE_PERST_B_LS
- Serial data input vector pair, PCIE_RX_P and PCIE_RX_N (4 wires each)
- Serial data output vector pair, PCIE_TX_P and PCIE_TX_N (4 wires each)

The pin assignment is made implicitly by placing the transceiver logic. The constraints defining the GTX placements in the UCF file force a certain pinout. Likewise, the placement of the reference clock's pins is implicitly set by constraining the position

of the clock buffer (pcieclk_ibuf). The UCF file in Xillybus' bundle contains guiding comments.

The UCF file must be edited so that the pin placements of these match those of the intended board.

4.3.5 Working with Virtex-5 PCIe boards

There are two groups of devices within the Virtex-5 family, each requiring a slightly different PCIe interface. To handle this simply, there are two different XCO files in the 'blockplus' subdirectory, only one of which should be used.

Accordingly, it's necessary to rename a file in that subdirectory as follows before building the PCIe core:

- For Virtex-5 LX or Virtex-5 SX: Rename pcie_v5_gtp.xco to pcie_v5.xco
- For Virtex-5 FX or Virtex-5 TX: Rename pcie_v5_gtx.xco to pcie_v5.xco

IMPORTANT:

The version of PCIe Block Plus generator should be 1.14, and definitely not 1.15. ISE 13.1 has the correct version for this purpose, but the one that arrives with ISE 13.2 will produce faulty code.

If a version other than ISE 13.1 is desired for the overall implementation, it's possible to generate the Verilog files with the correct version of PCIe Block Plus (included in ISE 13.1). The implementation of the entire project can then be done in the preferred version of ISE.

The UCF file has guiding comments on how the pins should be set up. The placement of the PCIe pins is implicit, and is forced by the constraint on the position of the GTP/GTX component.

A clock with a frequency of 100 MHz is expected on the PCIE_REFCLK wire pair. If a different clock is applied, the Xilinx PCIe Coregen core (defined by pcie_v5.xco) must be reconfigured to expect the real clock frequency. Additionally, a timing constraint must be updated, so that the TS_MGTCLK specification reflects the change.

4.3.6 Working with Kintex-7, Virtex-7 and Artix-7 boards (PCIe)

All FPGAs in the series-7 family have the same PCIe interface.

- A pair of differential wires for the reference clock, with the names PCIE_REFCLK_P and PCIE_REFCLK_N: A clock with a frequency of 100 MHz, which is derived from the PCIe bus clock (or connected directly), is expected on these wires.

  If a different clock is applied, the PCIe block (defined by pcie_k7_vivado.xci or similar in the demo bundle) must be reconfigured to expect the real clock frequency. This file appears in the project's list of sources. Such a change may also involve changes in the timing constraints. Please refer to the example XCF file, generated by Xilinx' tools.

  If ISE is used, Xilinx' PCIe core is defined by e.g. pcie_k7_8x.xco. The UCF file, rather than XDC, may need adjustment.

- The host's master bus reset on PCIE_PERST_B_LS
- Serial data input vector pair, PCIE_RX_P and PCIE_RX_N (8 or 4 wires each)
- Serial data output vector pair, PCIE_TX_P and PCIE_TX_N (8 or 4 wires each)

The pin assignment is made implicitly by placing the transceiver logic. The constraints defining the GTX placements in the UCF/XDC file force a certain pinout. Likewise, the placement of the reference clock's pins is implicitly set by constraining the position of the clock buffer (pcieclk_ibuf). The UCF / XDC file in Xillybus' demo bundle contains guiding comments.

The UCF/ XDC file must be edited so that the pin placements of these match those of the intended board.

4.3.7 Working with Ultrascale and Ultrascale+ boards (PCIe)

All of these FPGAs have the same PCIe interface.

- A pair of differential wires for the reference clock, with the names PCIE_REFCLK_P and PCIE_REFCLK_N: A clock with a frequency of 100 MHz, which is connected directly to the PCIe bus' clock.

  If a different clock is applied, the PCIe block (defined by pcie_ku_vivado.xci or similar in the demo bundle) must be reconfigured to expect the real clock frequency, and the timing constraint for this clock must be updated in xillydemo.xdc. These files appears in the project's list of sources.

- The host's master bus reset on PCIE_PERST_B_LS
- Serial data input vector pair, PCIE_RX_P and PCIE_RX_N (8 or 4 wires each)

- Serial data output vector pair, PCIE_TX_P and PCIE_TX_N (8 or 4 wires each)

The pin assignment is made implicitly by placing the transceiver logic. The constraints defining the GTX placements in the XDC file force a certain pinout. Likewise, the placement of the reference clock's pins is implicitly set by constraining the position of the clock buffer (pcieclk_ibuf). The XDC file in Xillybus' demo bundle contains guiding comments.

The XDC file must be edited so that the pin placements of these match those of the intended board.

4.3.8 Working with Versal ACAP boards (PCIe)

These FPGAs have the following PCIe interface:

- A pair of differential wires for the reference clock, with the names PCIE_REFCLK_P and PCIE_REFCLK_N: A clock with a frequency of 100 MHz, which is connected directly to the PCIe bus' clock.

  If a different clock is applied, the PCIe controller in the CPM block (which is inside the CIPS IP in the pcie_versal block design) must be reconfigured to expect the real clock frequency.

- Serial data input vector pair, PCIE_RX_P and PCIE_RX_N (8 wires each)

- Serial data output vector pair, PCIE_TX_P and PCIE_TX_N (8 wires each)

The host's master bus reset is connected directly to PMC MIO 38. If another MIO pin is used for this signal, the CIPS IP must be configured accordingly (see Xillybus' tutorial page for more information).

The XDC doesn't contain any constraints that are related to the PCIe block. Both the timing constraints and the pin placement constraints are supplied by the CIPS IP implicitly. Note that the pin placements can't be moved, since the CPM is used for the PCIe interface.

The procedure for changing the PCIe block's parameters is different from other FPGAs: The PCIe part is implemented as a block design. In this block design, there is a block that is named pcie_block_support. This block contains the transceivers and clock resources that are used by the PCIe block. It is therefore required to update pcie_block_support after changing the number of lanes or the link speed. It's not enough to only update pcie_block.

The method for updating pcie_block_support is to delete this block and let Vivado regenerate it. This way, the block is created with the updated parameters of the PCIe block.

It is recommended to create a visual copy of the block design before starting this procedure. This will be helpful, because it is required to reconstruct the same block design with the updated pcie_block_support block.

The steps for this procedure are as follows:

- Delete the pcie_block_support block and all its external ports. This means to delete ports that are not connected to anything (e.g. "pcie_mgt"), and also delete ports that are connected to a block (e.g. "m_axis_cq_0").
- Delete the connection of the reset signal between versal_cips_0 and pcie_block.
- In response to the previous step, Vivado should suggest "Run Block Automation". Click on that that suggestion. Vivado will open pop-up windows in response. Verify that the PCIe parameters are correct, and click "OK". Note that Vivado will also suggest "Run Connection Automation", however this option is not sufficient.
- Vivado will add a new pcie_block_support block and make several connections.
- Remove the external port that is related to "sys_reset". Instead, connect versal_cips_0's pl_pcie0_resetn to the sys_reset inputs of two blocks: pcie_block and pcie_block_support. After doing this, "sys_reset" is connected like it was before.
- Select all pins of the PCIe block that aren't connected to anything (it's possible to use CTRL-click for this purpose). Make these ports external, by using right-click $>$ Make External. Vivado will create ports for all pins. The name of each external port will be like the net's name, with a "_0" suffix added.

4.3.9 Working with XillyUSB

XillyUSB can be used on other boards that have an SFP+ interface. In this case it's just a matter of setting the design's constraints to use the MGT that is wired to the SFP+ connector.

The board should also supply a 125 MHz reference clock with low jitter for the MGT. Despite the requirement in the USB specification, Spread Spectrum Clocking (SSC) should *not* be enabled (if such option exists): The MGT doesn't lock properly on the received signal if an SSC reference clock is used.

For custom boards, it's recommended to refer to the sfp2usb module's schematics, as the pins of the SFP+ connector are connected directly to the FPGA's MGT. It is optional to swap the SSRX wires, as done on the sfp2usb module. This is recommended only if it simplifies the PCB design.

Swapping the SSTX wires is also possible, if desired. This requires editing the *_frontend.v file so that the polarity of the transmitted bits is reversed, and hence compensates for the wire pair swap. Note that there's a good chance that the USB connection will operate properly even without this edit, since the USB specification requires the link partners to work properly even with a polarity swap. It's however recommended to not rely on this.

4.4 PRSNT pins for indicating the number of PCIe lanes

According to the PCIe spec, there is one or more pins on the PCIe connector, which indicate the presence of the peripheral in the PCIe slot, as well as the number of lanes. These are the PRSNT pins. Most development boards have DIP switches for adjusting how many lanes the host is informed about, by virtue of these pins.

The typical default setting of these pins is the maximal number of lanes that is possible with the board. This setting usually works, even if less lanes are actually used. This is because an initial negociation between the host and the peripheral (which is required by the PCIe spec) ensures the correct detection of the actual number of lanes.

Please refer to your board's reference manual about how to set these DIP switches. It's important not to set these DIP switches to less lanes than are actually used, since some hosts may ignore lanes as a result of a faulty setting.

4.5 Changing the number of PCIe lanes and/or link speed

4.5.1 Introduction

IMPORTANT:

Changing the link's parameters may require adjustments in the timing constraints. Failing to pay attention to this issue can lead to a PCIe link that works, but in an unreliable manner.
Always be sure to have properly adjusted the timing constraints, if necessary, after making changes. This topic is detailed below.

Xillybus' FPGA demo bundles are typically set to the maximal number of lanes avail-

able on the intended board, and a link speed of 2.5 GT/s (Gen1).

The rationale is that if an FPGA board fits into the PCIe connector of a motherboard, one can expect that all lanes will be used in the connection with the host. On the other hand, in almost all cases, the bandwidth that is achieved by these lanes is higher than the Xillybus IP core may utilize, even with 2.5 GT/s, and it's hence pointless to set a higher link speed.

As the PCIe specification requires a fallback capability to lower speeds from all bus components that are involved, picking 2.5 GT/s ensures a uniform behavior on all motherboards.

It's however often desired to change the number of lanes and their link speed, in particular when using the Xillybus IP core on a custom board. Less lanes with higher link speed is a common requirement.

The Xillybus IP core relies on Xilinx' PCIe block for the low-level interface with the PCIe bus. Accordingly, the IP core works properly as long as Xilinx' PCIe block operates properly, regardless of the number of lanes or their link speed.

If the PCIe block is configured with a low number of lanes, combined with a link speed, such that its bandwidth capability is lower than Xillybus IP core's, it will still work properly. In this case, the aggregate bandwidth offered by Xillybus' streams approximately equals the bandwidth limit imposed by the settings of the PCIe block. It's a question of what becomes the bottleneck.

4.5.2 The work procedure

For Versal ACAP FPGAs, please refer to section 4.3.8. The description below is intended for all other FPGAs.

In principle, changing the number of lanes and/or the link speed consists of making changes in the configuration of the PCIe block as desired. However there are a few issues to pay attention to:

- The modification may influence other parameters of the PCIe block, that may cause it to fail operating correctly. Among others, there's a bug in Xilinx' GUI tool to watch out for, as detailed below.
- The modification may change the frequencies of the clocks driving the PCIe block (the PIPE clocks), and hence requires changes in timing constraints.
- The said change in clock frequencies may also require changes in Verilog code

that supports the PCIe block (the instantiation of the pipe_clock module, where applicable), or else the PCIe block will not function.

The stages are hence as follows:

1. Make a copy of the XCO or XCI file on an active project (i.e. after Vivado or ISE has upgraded the IP as necessary). This will allow comparing the changes with a diff tool afterwards, and spot unwanted changes, if such occur.
2. Open the IP of the PCIe block in Vivado (or ISE) for configuration (with Versal FPGAs, open the CPM unit in the CIPS IP, which is the only block in the pcie_versal block design).
3. Change the number of lanes and/or Maximum Link Speed as desired, while paying attention not to change the (AXI) interface width. If it's possible to avoid changing the (AXI) Interface Frequency by choosing a combination of lanes and link speed that is adequate for the task, that is preferable.
4. After making the desired changes, verify that the Vendor ID and Device ID haven't changed (neither the Subsystem counterparts). Some revisions of Vivado may reset some parameters to their defaults as a result of unrelated modifications (this is a bug).
5. Confirm the changes (typically click "OK" at the bottom of the dialog box). There is no need to generate output products, if this is suggested following the confirmation.
6. Compare the updated and previous XCO or XCI files with a textual diff tool, and verify that only relevant parameters have changed. More on this below.
7. Adjust the signal vector width of PCIE_* in xillybus.v and xillydemo.v/.vhd, so they reflect the new number of lanes.
8. Adjust the PIPE clock module's instantiation if necessary, as described in paragraph 4.5.3 below.
9. Adjust timing constraints if necessary, as described in paragraph 4.5.4 below.
10. Update the PIPE clock module, as explained in paragraph 4.5.5.

The three last steps are not required when working with Ultrascale and later.

When comparing new and old XCI files, `PARAM_VALUE.Device_ID` should get special attention, as it's often changed accidentally.

The differences in the parameters of the XCI files should match those desired. This is a short list of possible parameters for which changes are acceptable in accordance with the changes made in Vivado. The names of the parameters should be taken with a grain of salt, as different revisions of Vivado (and hence different revisions of the PCIe block) may represent the attributes of the PCIe block with different XML parameters.

- Related to the number of lanes:
  - `PARAM_VALUE.Maximum_Link_Width`
  - `MODELPARAM_VALUE.max_lnk_wdt`
- Related to the link speed:
  - `PARAM_VALUE.Link_Speed`
  - `PARAM_VALUE.Trgt_Link_Speed`
  - `MODELPARAM_VALUE.c_gen1`
  - `MODELPARAM_VALUE.max_lnk_spd`
- Related to the the interface frequency. A change in these parameters is a strong indication that the stages in paragraphs 4.5.3 and 4.5.4 are necessary.
  - `PARAM_VALUE.User_Clk_Freq`
  - `MODELPARAM_VALUE.pci_exp_int_freq`

4.5.3 Has the PIPE frequency changed?

When working with Ultrascale FPGAs and later, the considerations and actions below are unnecessary, as their PCIe block supplies the timing constraints as an integral part of the IP itself. Same goes for the PIPE module.

As for other FPGA families, it's important to verify that the PIPE clock settings are correct as follows:

Generate an example project for the PCIe block **after the changes**, and run a synthesis of that project. In Vivado, this is typically done by right-clicking the PCIe block in the project's source hierarchy and select "Open IP Example Design...". After selecting a location for the design, and it has been generated, launch the synthesis by clicking "Run Synthesis" at the left column.

Next, obtain the PIPE clock module's instantiation parameters in the synthesis report (in Vivado, it's found as something like pcie_example/pcie_example.runs/synth_1/runme.log). In this report, search for a segment like:

```
INFO: [Synth 8-638] synthesizing module 'example_pipe_clock' [...]
Parameter PCIE_ASYNC_EN bound to: FALSE - type: string
Parameter PCIE_TXBUF_EN bound to: FALSE - type: string
Parameter PCIE_CLK_SHARING_EN bound to: FALSE - type: string
Parameter PCIE_LANE bound to: 4 - type: integer
Parameter PCIE_LINK_SPEED bound to: 3 - type: integer
Parameter PCIE_REFCLK_FREQ bound to: 0 - type: integer
Parameter PCIE_USERCLK1_FREQ bound to: 4 - type: integer
Parameter PCIE_USERCLK2_FREQ bound to: 4 - type: integer
Parameter PCIE_OOBCLK_MODE bound to: 1 - type: integer
Parameter PCIE_DEBUG_MODE bound to: 0 - type: integer
```

The parameters in this report must match those in the instantiation of pipe_clock, as they appear in xillybus.v, which is of the form:

```
pcie_[...]_pipe_clock #
   (
    .PCIE_ASYNC_EN                  ( "FALSE" ),
    .PCIE_TXBUF_EN                  ( "FALSE" ),
    .PCIE_LANE                      ( 6'h08 ),
    .PCIE_LINK_SPEED               ( 3 ),
    .PCIE_REFCLK_FREQ                 ( 0 ),
    .PCIE_USERCLK1_FREQ               ( 4 ),
    .PCIE_USERCLK2_FREQ               ( 4 ),
    .PCIE_DEBUG_MODE                  ( 0 )
    )
   pipe_clock
     (
       [ ... ]
      );
```

The three parameters to compare are `PCIE_LINK_SPEED`, `PCIE_USERCLK1_FREQ` and `PCIE_USERCLK2_FREQ`, which must match. If they do (as shown in the example), all is set correctly, including the timing constraints. If not, two actions must be taken:

- The instantiation parameters in xillybus.v must be updated to match those in the example project's synthesis report.
- The timing constraints must be adapted to the example project's. This is more difficult, because failing to do this correctly doesn't necessary cause a problem immediately, but may impact the design's reliability.

If the `PCIE_LANE` parameter in xillybus.v is larger than the example project's, there is

no problem leaving it that way, and it's often easier to do so.

4.5.4 Adapting the timing constraints

It's mandatory to adjust the timing constraints to reflect changes in the clocks of the PCIe block, if such have occurred.

As the constraints depend on the chosen FPGA as well as Vivado's revision, it may be somewhat difficult to get this done correctly. Avoiding this adjustment is the main motivation for attempting to keep the PIPE clock's frequency unchanged by selecting a combination of link speed and number of lanes (when possible). However even if the PIPE clock's frequency remains unchanged, updating the constraints may still be necessary.

Once again, when an FPGA of the Ultrascale family (and later) is used, there is no need to deal with timing constraints, because the IPs of their PCIe block handle this internally.

In order to adjust the timing constraints, first find the constraints of the example project. With Vivado, it's typically as a file of the form `example.srcs/constrs_1/imports/example_design/xilinx_*.xdc`.

It's highly recommended to generate one example project with the PCIe block's setting before the changes in number of lanes and/or link speed, and one example project after these changes. A simple diff between the two example projects' constraint files gives the definite answer to whether adaption of the constraints is required, and if so, in what way.

Compare the constraint file's "Timing constraints" section with those in xillydemo.xdc. The example project selects logic elements by their absolute position in the logic hierarchy, so some editing is necessary. For example, suppose a timing constraint like this in the example project:

```
set_false_path -to [get_pins {pcie_vivado_support_i/pipe_clock_i/
pclk_i1_bufgctrl.pclk_i1/S0}]
```

In xillydemo.xdc it should be written as:

```
set_false_path -to [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
```

The main differences are with the relative paths used in Xillybus' constraints. There

might also be other slight differences, as some constraints are necessary with earlier revisions of Xilinx' tools, and become superfluous with later ones.

After making the changes in the timing constraints, it's important to verify that they took effect on logic by reviewing the timing report after the design's implementation.

Finally, it's worth explaining the following two constraints, which are present in xillydemo.xdc of some demo bundles:

```
set_case_analysis 1 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
set_case_analysis 0 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S1]
```

These constraints are required for Gen1 PCIe blocks, when using quite old revisions of Vivado, as explained in Xilinx AR #62296. Hence they may be omitted with recent Xilinx tools.

4.5.5 Updating the PIPE clock module

As mentioned above, this step isn't required for Ultrascale FPGAs and later.

In some cases, in particular when increasing the link speed from or to 2.5 GT/s (Gen1), it's required to update the pcie_*_vivado_pipe_clock.v file, which resides in the vivado-essentials/ directory.

This file is generated automatically as part of the initial project setup by virtue of executing xillydemo-vivado.tcl. It may change slightly depending on the configuration of the PCIe block, in particular if it's limited to 2.5 GT/s or not.

The recommended way is to regenerate the Vivado project with the xillydemo-vivado.tcl script. Namely, start from a fresh demo bundle, and copy the files that were changed during the previous stages into it:

- The XCI file of the PCIe block
- xillydemo.xdc
- The Verilog / VHDL files that were edited for adapting to the new number of lanes and link speed.

With these files copied into the new demo bundle, generating the project with the xillydemo-vivado.tcl script ensures that the PIPE clock module is in accordance with

the settings of the PCIe block, and also that the project doesn't depend on any leftover files from before the change.

Alternatively, update the pcie_*_vivado_pipe_clock.v file from the example project that was created in paragraph 4.5.3. The file to be used has exactly the same name as the one in vivado-essentials/, and is typically found deep into the example project's file hierarchy. Copy this file into vivado-essentials/ (overwriting the existing one).

4.6 Changing the FPGA part number

When migrating from one FPGA family to another, it's necessary to start from a different demo bundle. There are differences (which are sometimes subtle) that relate to the PCIe block, (as well as the MGT block with XillyUSB). These differences require a different Xillybus IP core as well as different wrapper modules.

Attempting to just change the project's part in Vivado / ISE may result in a project with errors during its implementation. Even if the implementation is successful, the logic may not work, or work unreliably.

However when remaining within the same FPGA family, changing the part number is often sufficient (along with the considerations mentioned above regarding pin placement and constraints).

It's important to note that for some FPGA families (Ultrascale in particular), the position (site) of the PCIe block in the logic fabric is an attribute of the PCIe block itself, and it may therefore require modification. Moreover, each specific FPGA, and each specific package, has its own set of valid sites. Hence the change of FPGA may reset the PCIe IP's attributes, if the site that is selected for the PCIe block doesn't exist on the new FPGA.

Vivado's reaction to an invalid site in the PCIe block's position (site) is quite destructive. If this is the case, the "upgrade" of the PCIe block (the operation which is always required to unlock the IP after changing the FPGA) results in resetting several attributes of the PCIe block, to arbitrary values. While doing so, a Critical Warning like the following is generated:

```
CRITICAL WARNING: [IP_Flow 19-3419] Update of 'pcie_ku' to current project
options has resulted in an incomplete parameterization. Please review the
message log, and recustomize this instance before continuing with your design.
```

Attempting an implementation of the project without paying attention to this issue is completely pointless, and leads not just to a large number of misleading warnings,

Critical Warnings and possibly errors, but the result, if any, will be far from being functional.

The solution is to assign a PCIe block site that is valid on the new FPGA, before changing the part number attribute for the project. This might require editing the XCI file manually if there is no common site between the FPGA part before and after the change (because in this case, it will not be possible to make this change in the GUI, which limits the setting to the sites allowed on the current FPGA part).

5

Troubleshooting

5.1 Errors during implementation

Slight differences between releases of Xilinx' tools sometimes result in failures to run the implementation for creating a bitfile.

If the problem isn't solved fairly quickly, please seek assistance through the email address given at Xillybus' web site. Please attach the output log of the process that failed, in particular around the first error reported by the tool. Also, if custom changes were made in the design (i.e. diversion from the demo bundle) please detail these changes. Also please state which version of Vivado (or ISE) was used.

5.2 PCIe Hardware problems

Normally, the PCIe card is detected properly by the host's BIOS and/or operating system, and the host's driver launches successfully.

On most PC computers, the BIOS briefly displays a list of detected peripherals early in the boot process. When the Xillybus interface is detected successfully, a peripheral with vendor ID 10EE and device ID EBEB appears on the list.

As for the operating system's detection of the card, please refer to one of these two documents, whichever applies:

- Getting started with Xillybus on a Linux host
- Getting started with Xillybus on a Windows host

The failure to detect the card (or a failure in the computer's boot process) is not related to the Xillybus IP core, which relies on Xilinx' PCIe IP core for interfacing with the bus.

At first, it's recommended to verify the following:

- The bitstream was already loaded into the FPGA when the computer was powered on (or soon enough after it was powered on, in terms of the PCI-SIG specification).
- The pinouts of the PCIe wires, including the reference clock are correct (this can be verified in the placement report).
- The board supplies the correct reference clock to the FPGA.

If the problem isn't spotted immediately, it's recommended to attempt the sample project for PCIe that came with the board. This may reveal wrong jumper settings and possibly defective hardware.

If the card is detected with this sample, but not with Xillybus, it may be helpful to compare the pinouts of the two designs. If they are equal, the next step is comparing the attributes of the Xilinx' PCIe cores by invoking the IP GUI for each (double-clicking the XCI / XCO element in the Project Manager with each of the projects open).

The following configuration elements may need adjustment:

- The frequency of the reference clock (may appear as "Interface Frequency" in the GUI).
- The base class and sub class (not likely, but some relatively old PC computers have failed the boot process if the class was considered unknown).
- Any other attribute that is configured differently, except for the base address register settings, vendor ID, device ID and interrupt settings, which should not be altered.

If the problem remains, please seek assistance through the email address given at Xillybus' web site.